Feature-Based Surface Parameterization and Texture Mapping

Eugene Zhang, Konstantin Mischaikow and Greg Turk
Georgia Institute of Technology

Surface parameterization is necessary for many graphics tasks: texture-preserving simplification, remeshing, surface painting, and pre-computation of solid textures. The stretch caused by a given parameterization determines the sampling rate on the surface. In this paper, we propose an automatic parameterization method that segments a surface into patches that are then flattened with little stretch.

We observe that many objects consist of regions of relative simple shapes, each of which has a natural parameterization. Therefore, we propose a three-stage feature based patch creation method for manifold mesh surfaces. The first two stages, genus reduction and feature identification, are performed with the help of distance-based Morse functions. In the last stage, we create one or two patches for each feature region based on a covariance matrix of the feature's surface points.

To reduce the stretch during patch unfolding, we notice that the stretch is a 2×2 tensor which in ideal situations is the identity. Therefore, we propose to use the *Green-Lagrange tensor* to measure and to guide the optimization process. Furthermore, we allow the boundary vertices of a patch to be optimized by adding *scaffold triangles*. We demonstrate our feature identification and patch unfolding methods for several textured models.

Finally, to evaluate the quality of a given parameterization, we propose an image-based error measure that takes into account stretch, seams, smoothness, packing efficiency and visibility.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling – Geometric algorithms, languages, and systems

General Terms: Algorithms

Additional Key Words and Phrases: Parameterization, Surface Segmentation, Texture Mapping

1. INTRODUCTION

Surface parameterization is a well-studied problem in computer graphics. In general, surface parameterization refers to segmenting a 3D surface into one or more patches and unfolding them onto a plane. Borrowing terminology from mathematics, this is often referred to as creating an *atlas* of *charts* for a given surface. This process is necessary for many graphics applications in which properties of a 3D surface (colors, normals) are sampled and stored in a texture map. The quality of the parameterization greatly affects the quality of subsequent applications. One of the most important quality measurements is stretch. When unfolding a surface onto a plane, stretching occurs if the surface contains highly spherical or hyperbolic regions, which have very different geometric structures from that of Euclidean spaces. High stretch in a parameterization results in uneven sampling.

We observe that objects can be decomposed into a set of "simple" shapes that roughly approximate cylinders and cones, flat disks and spheres. Cylinders, cones and planes are developable surfaces, which are Euclidean by nature. Unfolding them result in little stretch. In this paper, we propose to use distance-based Morse functions to divide a closed manifold into feature regions, each of which is similar to one of the simple shapes. Figure 1 shows the result of our algorithm on the bunny surface. The left portion shows such a feature decomposition, while the right portion shows the surface parameterization obtained based on this decomposition.

Existing patch unfolding techniques are often divided into two stages: initial patch layout to achieve some objective such as conformal mapping, followed by interior vertex optimization based on some geometric stretch. We observe

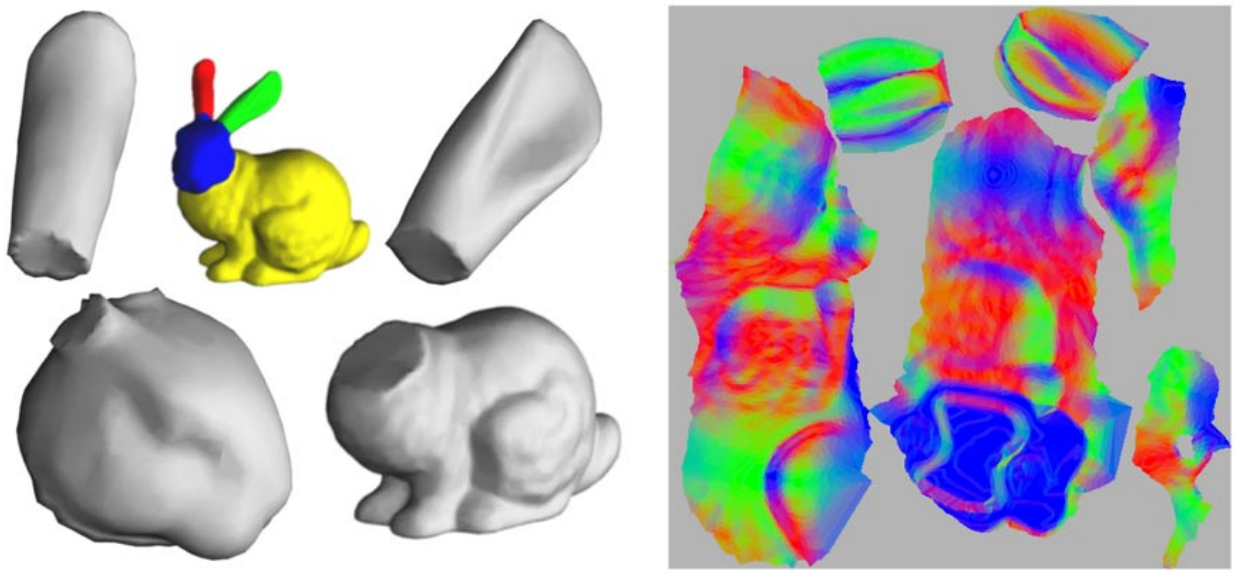

Fig. 1. The feature regions (left) and the unfolded patches (color coding the surface normal, right) based on our algorithm.

that an ideal surface parameterization between a patch and its textural image is an isometry, that is, distance-preserving between any two points on the surface and their images in texture space. Green-Lagrange deformation tensor has the property that it measures anisotropic stretch faithfully and penalize undersampling more severely than oversampling. In addition, it can be seen as a balance between area-preserving mapping and conformal mappings. We use this metric to guide the vertex optimization process for patch unfolding. In addition, we use what we call *scaffold triangles* to convert the original boundary vertices into "interior" vertices, which can then be freely moved around within the same optimization framework. This is a new way of creating non-convex patches that may even have holes.

The last stage of building surface parameterization is patch packing. The goal of patch packing is to lay each unfolded patch into a rectangular domain without overlapping with other patches. The packing ratio measures the percentage of this domain being covered by a patch. Higher packing ratio indicates better use of the textural space. Existing methods use a horizontal-sweeping fashion to insert new patches. To prevent placing new patches over existing patches, "horizon" of the used space is kept track of. For surface parameterization algorithm in which large patches are present, there are often a large area of unused space below the horizon. We propose to a new patch packing algorithm to allow spaces to be used even if they are below a horizon. Furthermore, we allow patches to be inserted either along its long axis or its short axis. The combination of the two methods improve the packing ratio.

The remainder of the paper is organized as follows. In Section 2 we review existing surface parameterization methods. Then, we present our feature-based patch creation method in Section 3, followed by our new balanced stretch metric in Section 4.1, boundary vertex optimization technique in Section 4.2, and our new packing algorithm 5. Section 6 shows results from our technique, and Section 7 provides a summary and discusses future work.

2. PREVIOUS WORK

There has been a considerable amount of recent work in the graphics community on building surface parameterization by unfolding a polygonal surface into planar patches. Much of the motivation for this is for *texture mapping*, the mapping of pixels from a rectangular domain (the *texture map*) onto a surface that is described by a collection of polygons. The patch creation problem is to subdivide the given surface into a (hopefully small) number of patches that are then flattened onto the plane and arranged in the texture map. Uses for such patch creation methods include

surface painting [Hanrahan and Haeberli 1990], fast rendering of procedural textures [Perlin 1985; Carr and Hart 2002], applying photographed color variations onto digitized surfaces [Cignoni et al. 1998], and creating normal maps from detailed geometry [Sander et al. 2001]. These same patch creation methods may also be used for *remeshing*, that is, for creating a new mesh from the original surface [Alliez et al. 2002]. Remeshing can be used to improve the triangle shapes, to vary the triangle size according to curvature details, and to induce semi-regular tessellations. Recently, octrees have been used to store colors in 3D for surface texturing without any parameterization [Benson and Davis 2002; DeBry et al. 2002]. Unfortunately these octree techniques are not yet supported by graphics hardware.

2.1 Patch Creation

There are two common approaches to the patch creation problem. The first of these is to find a single cut of the surface that makes the modified surface topologically equivalent to a disk [Piponi and Borshukov 2000; Gu et al. 2002; Sheffer and Hart 2002; Erickson and Har-Peled 2002; Alliez et al. 2002]. This cut surface is unfolded into a single planar patch. This approach has the virtue of creating as few seams as possible, but will often introduce large stretch between the patch and the surface. Such stretching is undesirable because different portions of the surface are represented using quite different amounts of color detail, as measured in pixel resolution in the texture map.

The other major approach is to divide the surface into a collection of patches that can be unfolded with little stretch [Eck et al. 1995; Lee et al. 1998; Sander et al. 2001; Lévy et al. 2002; Sorkine et al. 2002]. Though stretch is minimized, this approach creates seams between the patches. These seams cause problems when creating textured images of the surface because the color variation across the seams must be treated with extreme care or the seams will be noticeable. Some methods create small disk-like patches [Eck et al. 1995; Lee et al. 1998; Sander et al. 2001], while others attempt to create large patches that match the features of an object [Lévy et al. 2002; Sorkine et al. 2002]. Our own work takes this latter approach. We cut the surface into multiple patches, but according to the large geometric features of the surface. For example, we would like to recognize the head and limbs of an animal as important features and to create patches that respect these features. The work of Lévy et al. [2002] has similar goals, although their feature-based patching method is quite different than our own.

The term *geometric feature* often have its context. For surface reconstruction and mesh simplification algorithm, features are often defined in terms of local curvature. This is fine because high curvature regions are exactly what these applications are trying to preserve. On the other hand, surface parameterization algorithms incur higher stretch on a smooth surface with long thin protrusions than a noisy surface with little protrusions. In this work, we define geometric features as large protrusions, and propose to find and segment these features with some surface distance-based functions.

2.2 Patch Unfolding

There have been many patch unfolding techniques. The classical approach treats the patch unfolding problem as finding the minimal of some functional that measures the difference between a parameterization with isometry [Eck et al. 1995; Floater 1997]. First, the boundary vertices are assigned initial positions (usually on a circle). Then the parameterization for interior vertices is determined by solving a large linear system or through nonlinear optimization. Others have used stretch measures such as the Green-Lagrange deformation tensor [Maillot et al. 1993] and a variant of Dirichlet energy [Hormann and Greiner 1999]. Sander et al. [2001] defined a geometric metric based on the average and maximal stretch in all direction of a triangle. They also proposed a post-processing nonlinear optimization step that improves the geometric stretch. As we describe later, Sander's patch optimization approach was an inspiration for our own work. To allow the boundary vertices of a patch to be free from the initial arbitrary assignment, Lévy et al. [2002] uses a least squares conformal mapping. Sander et al. [2002] allow boundary vertices to move while checking for global intersection. Lee et al. [2002] add layers of "virtual" boundaries as part of the edge springs to allow the patch boundaries to have a natural shape.

3. FEATURE-BASED PATCH CREATION

For a genus zero surface, our feature-based patch creation method is carried out in two stages. We first perform feature identification, in which geometric features (protrusions) are repeatedly found and separated from the rest of the surface. The resulting surfaces are also processed using feature identification until none of them have large geometric features. Second, we perform patch creation, in which each feature region is analyzed and divided into patches according to a covariance matrix of its surface points. For a genus $n(>0)$ surface, we use a third stage and modify the surface to be genus zero (*genus reduction*) before we begin feature identification.

For both genus reduction and feature identification, we build a surface-based Reeb Graph (the *exoskeleton*) that depends on the *average geodesic distance* introduced by Hilaga et al. [2001]. Since our goal is to create patches that are topological disks, we need to perform our operations in a topological consistent manner. For this purpose, we use surface region growing for all three stages: genus reduction, feature identification and patch creation. By starting from an initial triangle, we grow a region by adding one triangle at a time until the whole surface has been covered or when some other stopping criterion has been met.

Next, we will describe the average geodesic distance function and the Reeb Graph that it induces.

3.1 The Average Geodesic Distance

The *average geodesic distance* was introduced by Hilaga et al. [2001] for the purpose of shape matching. This is a function $A(\mathbf{p})$ that takes on a scalar value at each point $\mathbf{p}$ on the surface of an object S. Let $g(\mathbf{p},\mathbf{q})$ be the geodesic distance between two points $\mathbf{p}$ and $\mathbf{q}$ on the surface. Then the average geodesic distance is defined as follows:

$$A(\mathbf{p}) = \int_{\mathbf{q}\in S} g(\mathbf{p},\mathbf{q}) d\mathbf{q} \tag{1}$$

For our application, we use a slightly different function defined as follows:

$$G(\mathbf{p}) = \int_{\mathbf{q}\in S} g^2(\mathbf{p},\mathbf{q}) d\mathbf{q} \tag{2}$$

$G(\mathbf{p})$ is smoother than $A(\mathbf{p})$ because surface points that are further away from $\mathbf{p}$ are assigned more weights. Finally, we normalize function G to obtain AGD, the *normalized average geodesic distance function*:

$$AGD(p) = \frac{G(\mathbf{p})}{\min_{\mathbf{q}\in S} G(\mathbf{q})} \tag{3}$$

This function has several useful properties. First, its value measures how "isolated" a point is from the rest of the surface. Second, we observe its local maxima coincide with the tips of geometric features. Third, it is scale invariant and can even be used to compare features from different shapes. Figure 2 (left) shows a polygonal model of a dinosaur, color-coded according to AGD. The red region on the dinosaur's belly signifies that points in this region have a low values of AGD. Higher values adjacent to this middle region are green and then blue. The colors then cycle repeatedly through red, green, blue. Note that the tips of the large features of this object (legs, horns, tail) are marked by local maxima of AGD.

For a genus zero surface, we use AGD to identify and measure its geometric features. Each local maxima of this function is the tip of a geometric feature. Larger values at the local maxima signifies larger geometric features. In practice, we consider any local maximum at a point $\mathbf{p}$ to be the tip of a geometric feature if $AGD(p)$ is larger than 2.0. In fact, we find that choosing any number in $[1.5, 2.0]$ as the threshold for minimal feature size produces reasonable results.

Computing the AGD function exactly would be quite costly. By following closely to the algorithm by Hiliga et al. [2001], we can quickly compute a satisfactory approximation of AGD. Briefly, the geodesic distances are not

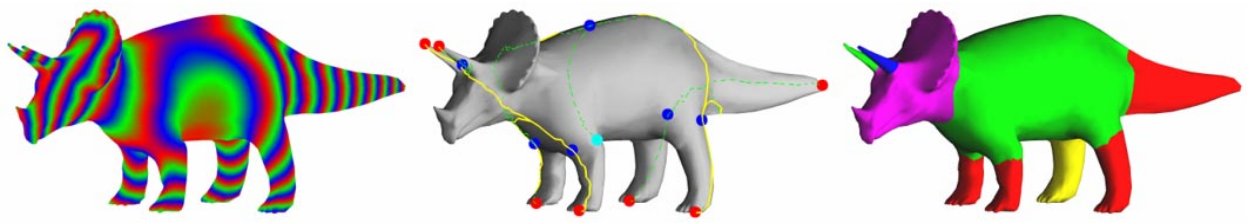

Fig. 2. The *average geodesic distance* (AGD) of the dinosaur model is color-coded in the left portion of this figure. The global minimum is underneath the belly, in red. Level sets are painted in repeated patterns of red/green/blue. Notice the levelset bands change topology and end on the tips of features. The middle portion shows the *exoskeleton* created by surface growing based on the AGD. The local maxima are shown with red spheres, and saddle points are at the blue spheres. Successive *critical points* are connected by surface paths shown in solid yellow (front) and dash green (hidden). The right portion shows the final surface segmentation into feature regions that were identified based on the exoskeleton.

calculated from all points **q**, but rather from a small number of evenly spaced points on the surface of S. We find the geodesic distances from each of these points to all other points efficiently using the fast marching method for surfaces of Kimmel and Sethian [1998].

3.2 Building Exoskeleton and Finding Features

To find the local maxima of AGD, we draw upon ideas from *Morse Theory* [Milnor 1963] and Reeb graphs. Using AGD, the average geodesic distance, we build a graph on the surface of the given object, and because it lies on the outside of the surface we call this graph the *exoskeleton*. The exoskeleton is much like a Reeb graph, but where a Reeb graph has one arc, the exoskeleton has a sequence of edges along a path on the surface. When properly designed and constructed, the exoskeleton reveals large geometric features for genus zero surfaces and contains loops that signify the location of handles for genus reduction of genus $n(>0)$ surfaces. We briefly review the related results from Morse Theory, first just for genus zero surfaces and later for any genus.

For a given smooth scalar function f on a surface S, the *critical points* of f are points in S with zero gradient. When the critical points are isolated and non-degenerate, f is called a *Morse Function*. There are three types of critical points, minima, maxima and saddle points. AGD is in general not a Morse function, but we can build a Morse function on S that has the same set of local maxima as AGD. This function is implicitly built by region growing over S in the increasing order of AGD.

Starting from the global minimum of AGD, we add one triangle at a time until the surface has been covered. Critical points occur when the topology of the region changes, or equivalently, when the topology of the boundaries changes. The advantage of adding one triangle at a time is to ensure that no more than one critical point can occur simultaneously. There are three types of critical points for a genus zero surface:

(1) Minima: at which one boundary starts. For our function, there is only one such point, the global minimum of the AGD.
(2) Maxima: at which one boundary vanishes. These are the tips of geometric features.
(3) Splitting saddle points: at which one boundary splits into two boundaries.

The front of the growing region moves over the surface in the order given by our Morse function. The regions of the surface that are swept out between pairs of critical points (not including these critical points) are homeomorphic to a cylinder without caps. If $AGD(\mathbf{p}) < AGD(\mathbf{q})$ for two critical points **p** and **q** at either end of such a cylinder, we will refer to **p** as the *parent critical point* of **q**. Similarly we will refer to **q** as the *child critical point* of **p**. For genus zero surfaces, each child critical point has a single parent. For surfaces with genus greater than zero, a child critical point may have one or two parents. If we connect every pair of parent-child critical points with paths on surface, we obtain a graph that we call the exoskeleton. The root of the exoskeleton is the minimum, and its internal nodes and leaf nodes are saddle points and maxima respectively. For genus zero surfaces the exoskeleton is a tree, but for higher

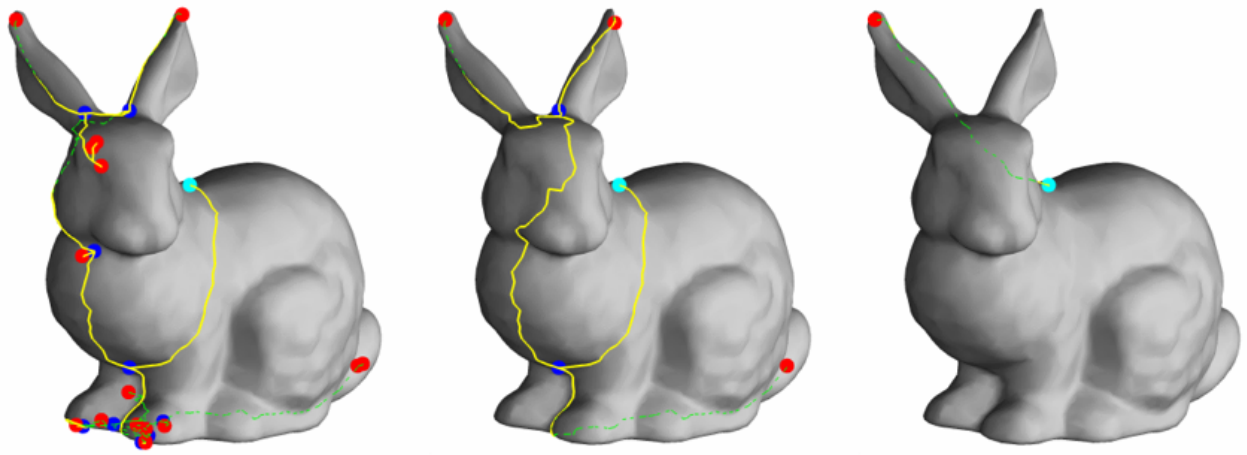

Fig. 3. This figure shows the exoskeletons of the bunny with different filtering constants. From left to right, the filtering constants α are: 1.0001, 1.2, and 2.5. We use 1.2 as our filtering constant for all test models.

genus surfaces the graph contains loops. Let N_{max} and N_{ss} be the number of the local maxima and splitting saddle points, respectively. For the case of genus zero surfaces we have:

$$N_{max} - N_{ss} = 1 \tag{4}$$

As we mentioned earlier, the exoskeleton is like the Reeb Graph that corresponds to the Morse function *AGD*. It reveals the geometric feature distribution over surface. The middle image of Figure 2 shows the exoskeleton of the dinosaur. Local maxima are highlighted with red spheres, while blue spheres indicate the location of splitting saddle points. The global minimum is marked with a light blue sphere on the belly. Successive critical points are connected by paths on surface, which are drawn in solid yellow (visible) and dash green (hidden). Note the local maxima coincide with the tips of geometric features (horns, feet, tail).

Since complex surfaces contain many small features, the exoskeleton can contain excessive numbers of local maxima and saddle points. This increases the subsequent processing time since the number of features is much more than we require. We only want the largest features, so we use a filtering scheme to weed out extra local maxima and splitting saddle points. During surface growing we alter the order in which triangles are added. To be more specific, let $\mathbf{t}$ be the triangle with the least AGD. If adding $\mathbf{t}$ causes a boundary to split, we look for other triangles that could be added without causing boundary split. If one of these triangles, $\mathbf{t}'$ satisfies:

$$AGD(\mathbf{t}') < \alpha AGD(\mathbf{t}) \tag{5}$$

where α is a global filtering constant, then we add $\mathbf{t}'$ instead of $\mathbf{t}$. In practice, we find $\alpha \in [1.2, 2.0]$ works well for all of our test models.

The left portion of Figure 3 shows the filtered exoskeleton with $\alpha = 1.0001$. Notice some excessive saddle points and maxima appear in the head and the pawns. The middle portion is obtained by using $\alpha = 1.2$. Notice how the local maxima that reveal large geometric structures remain (tip of ears, center of tail). Excessive filtering will result in a trivial exoskeleton if $alpha$ is too high ($\alpha = 2.5$), shown in the right portion of the same figure. This becomes a classical trade off between de-noising and over blurring.

Region growing on a genus $n(> 0)$ surface will incur a fourth type of critical point, the *merging saddle point*, at which two boundaries merge into one. A merging saddle point always indicates the presence of a handle. Unlike local maxima and splitting saddle points, it is topological by nature. Let N_{ms} be the number of merging saddle points, then

for a surface of genus n we have:

$$N_{ms} = n \tag{6}$$

$$N_{max} - N_{ss} + N_{ms} = 1 \tag{7}$$

The exoskeleton for a genus n surface contain exactly n merging saddle points. There are n loops that are mutually homologically inequivalent and form the bases of all loops in the graph. Later in Section 3.4 we describe how we use these loops for genus reduction, that is, converting a genus n surface into a genus zero surface.

3.3 Feature Identification

Once we find the tips of geometric features, we need to find a closed curve on the surface that separates the feature from the remaining body. We call this curve a *curve of separation*, γ. This is accomplished in two steps. First, we find a region of separation R, and then we construct γ from R.

To find the region of separation for a tip point p of a feature, we first calculate the function $f_{\mathbf{p}}(\mathbf{q})$ over the surface that is the geodesic distance from $\mathbf{p}$ to any point $\mathbf{q}$. We normalize this function to take on values in $[0,1]$, and then consider the regions bounded by iso-value curves of this function. Specifically, we divide the interval $[0,1]$ into k equal sections, and then using region growing from $\mathbf{p}$ we partition the surface into bands based on $f_{\mathbf{p}}$ in these intervals:

$$M_i := \{\mathbf{q} \in S | \frac{i-1}{k} \leq f(\mathbf{q}) \leq \frac{i}{k}\} \tag{8}$$

$$A_i := Area(M_i) \tag{9}$$

The area of this sequence of bands changes slowly on a feature, but the area changes abruptly when the feature joins another part of the surface. If we think of $\{A_i\}$ as a continuous function $A(x)$, then we can look at the behavior of this function to find the separation region. Along a perfect cylindrical feature, $A(x)$ is a constant function. In case of a cone, the function grows linearly. At regions where a protrusion intersects with the main body, $A(x)$ will have a sudden increase, and this will be the boundary of the feature. We find these increases by looking for maxima in the second derivative of $A(x)$. To eliminate small undulations in $A(x)$, we we first low-pass filter the function. Figure 4 illustrates this process.

Let m be the location where $A(m)$ takes on its maximum value. We define the region of separation as a set of points $\mathbf{q} \in S$:

$$R := \{\mathbf{q} \in S \mid m - \varepsilon \leq f_p(\mathbf{q}) \leq m + \varepsilon\} \tag{10}$$

We typically use $\varepsilon = 0.02$. We intentionally make R large for two reasons. First, poor triangulations may cause R to be non-separating, that is, not containing a closed loop that separates the tip $\mathbf{p}$ from the rest of the surface. Second, we would like more flexibility to allow the final curve of separation (within this region) to be short and smooth.

The topology of the separation region is usually a band, but in general it can be rather complex. The only guarantee is that this region indeed separates the feature from the rest of the surface. To find the separation curve γ based on R, we take the following steps:

1. Reduce the separation region into its skeleton. This is achieved by treating the separation region as a 2-complex (with boundary edges) and repeatedly performing "elementary collapses" [Kaczynski et al. 2001], in which triangles with at least one boundary edge are removed from the complex along with the boundary edges. At the end, all 2-cells (triangles) are removed and the 2-complex is reduced to a 1-complex. When there are multiple choices of boundary edges for collapse, we select the edge with the largest AGD, which tends to be further away from the feature tip p than other edges in the 2-complex. The resulting graph is the skeleton of region R. (This first step and the next are similar to the initial cut algorithm of Gu et al. [2002].)

2. Remove dangling edges. This is achieved by elementary collapses on the skeleton by removing dangling edges and their boundary vertices from the complex. This result in a collection of loops, one of which meets our requirement

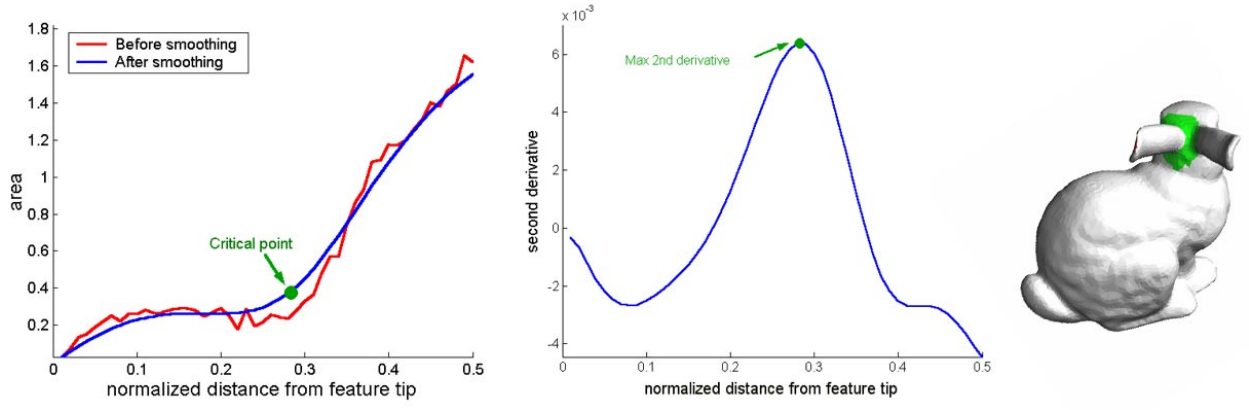

Fig. 4. This figure shows how we find the region of separation for one of the bunny's ears. In the left portion we graph $A(x)$, the area of the regions given by evenly spaced distances from the ear's tip (red), and the smoothed areas (blue). The second derivative of this function is then calculated (middle). The maximum corresponds to the place where the ear intersects with the head, shown in green on the right.

as the separation curve. The others fall into two categories: separation loops for small geometric feature inside the feature region, and separation loops for geometric features outside the feature region.

3. Eliminate separation loops outside the feature region. We perform region growing from the feature tip with the constraint that no triangles can be added that cross an edge in the loops computed from steps 1 and 2. Therefore the loops outside the feature region become unreachable.

4. Eliminate separation loops inside the feature region. The vertices on these loops in general have greater AGD values than the vertices on the separation loop. Therefore, they can be easily identified and discarded.

5. Shortening and smoothing the separation loop. The separation loop resulted from step 4 meet our topological requirement: they separate the feature region from the rest of the surface. To shorten and smooth the loop, we choose two vertices of the loop that are closest to the feature tip. We find the two paths from each of the two vertices to the feature tip, which divide the feature region into two disjoint regions. Within each region there is a shortest path between the two vertices. Together, these two shortest paths form a loop that is again a separation curve, except that it is shorter and smoother. By repeating this process twice, we obtain a closed loop that meet both our topological and geometric requirements.

We use the curve of separation to divide S into two surfaces, each with a boundary at the curve of separation. We eliminate these boundaries by "filling in" the two created holes with triangles. Some filler triangles can be seen where the head has been separated from the neck of the bunny in Figure 1. These filler triangles are flagged so that they have minimal effect on patch unfolding. They become what we call *scaffold triangles*, to be described later.

We repeat the feature identification process for the resulting surfaces until the original surface is divided a set of features regions and there are no more feature regions to be found. Figure 1 shows the result of this process on the bunny, with four regions having been created.

3.4 Genus Reduction

For a genus n surface ($n > 0$), a closed loop may not separate the surface into two disjoint connected components. (Loops with this properties are the so-called generators in Homology, which form an Abelian group with $2n$ generators.) Conceptually, the easiest way to think of how these loops arise is to imagine a hollow handle connected to the rest of the surface; one of the loops cuts across the handle and the other follows the handle. Observe that for the first type of loop there are two "passages" back to the rest of the surface. Our goal for genus reduction is to identify an appropriate loop across the handle and to use it to cut and block off the handle, thus reducing it to a protrusion. This

Fig. 5. This figure shows some blocking curves that are generated by our genus reduction algorithm for the happy buddha (left), the dragon (upper-right), and the feline (lower-right). Each blocking curve is a sequence of edges in the surface that form a closed loop. Notice they appear in intuitive places and are short and smooth.

process is repeated until a genus zero surface is generated. Erickson and Har-Peled [2002] proved that finding the minimal length cuts needed to turn a surface into a disk is NP-hard, so heuristics are used in practice to find cuts that are short in length. Genus reduction may be performed using a number of already existing techniques, including [Lazarus et al. 2001; Erickson and Har-Peled 2002; Gu et al. 2002; Sheffer and Hart 2002; Wood et al. 2002]. We choose to perform genus reduction using the same Morse Function that we use for feature identification.

At each step of genus reduction, we wish to find a short blocking curve (a curve that blocks one of the passages of at least one of the handles), cut the surface open along this curve, and fill the holes on both sides. Here again we make use of the exoskeleton that is given by AGD. First we locate all bases loops in the exoskeleton. Recall that a merging saddle point $\mathbf{q}_i$ signals that a handle has formed (Section 3.2). The start of a handle is a splitting saddle point $\mathbf{p}_i$. We trace back from the merging saddle point to the corresponding splitting saddle point along the paths in the exoskeleton. The two paths connecting the two saddle points is a basis loop in the exoskeleton.

For each basis loop, the splitting saddle point $\mathbf{p}_i$ is in general located near the ends of passages that connect the handle to the rest of the surface. We will create a nearby "blocking" curve for one of the passages. To find the curve, we use a region growing approach with a different Morse function and a different starting condition. First, the basis loop can be considered as two boundaries of a empty 2-cell. We perform region growing from this 2-cell based on the distance function from $\mathbf{p}_i$. Since the surface has handles, the region growing operation will find the merging saddle points of this new distance function. Let $\mathbf{q}_{i,new}$ be the first of such points, and region growing is stopped when this happens. Let R be the region that has been covered up to this point, and let $\mathbf{p}_{i,new}$ be the vertex in the basis loop that is closest to $\mathbf{q}_{i,new}$. Again, we can easily find two paths connecting $\mathbf{p}_{i,new}$ and $\mathbf{q}_{i,new}$ since basis loop divides R into two disjoint regions. The two connecting paths form a loop $l_{i,new}$ which is also a generator that may or may not be homologically equivalent to the basis loop. We use $l_{i,new}$ to be our blocking curve.

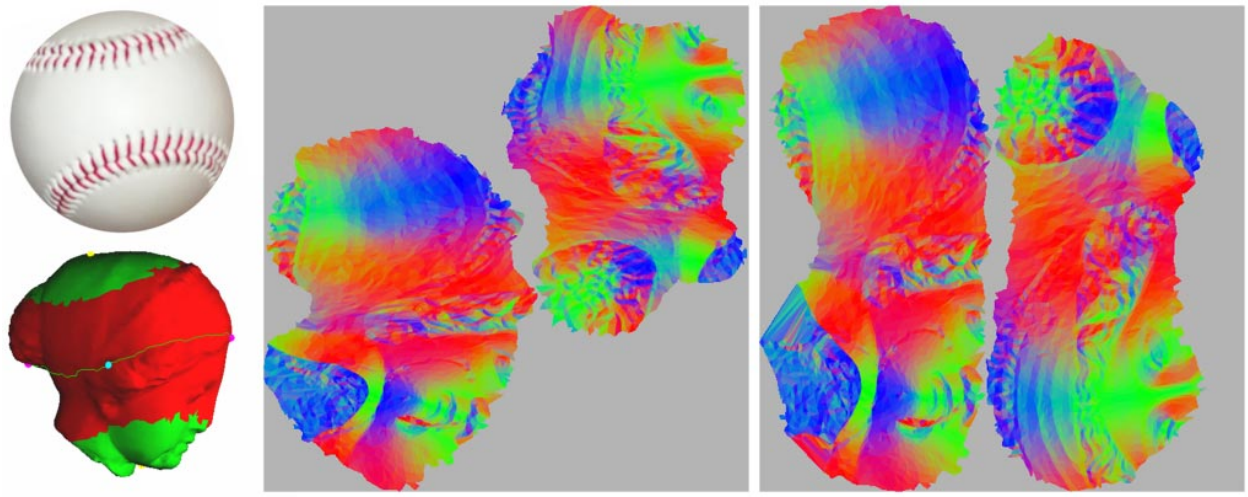

Fig. 6. This figure shows the "baseball" decomposition of the igea model (upper-right) and the corresponding normal maps (bottom row) from patch unfolding with different stretch metric: geometric stretch [Sander et al. 2001] (left) and Green-Lagrange tensor (Section 4.1, right).

We split the surface based on each blocking curves. Then, just as we did with the separating curves for features, we use filler triangles to patch over the two boundaries that the splitting creates. Each time that we find the shortest blocking curve, split the surface, and fill in the boundaries, we reduce the genus of the surface by one. We do this repeatedly until we have a genus zero surface. We perform this genus reduction *before* the feature identification process.

Figure 5 shows some blocking curves that are generated by our genus reduction algorithm for genus $n(>0)$ surfaces. These curves appear in intuitive places and they are short and smooth. These genus zero surfaces are ready for feature identification.

3.5 Patch Creation

Once the model has been divided into separate feature regions, we then need to create one or more patches from each feature. Some features such as long cylinders can be unfolded into a single patch, but others (such as nearly spherical features) are best divided into two patches. We classify each feature as one of three types: a sphere, a long thin ellipsoid, and a flat ellipsoid. Recall that a feature F is a collection of triangles. To classify the type of the feature, we follow the method of Gottschalk et al. [1996] that begins by thinking of F as having been sampled "infinitely densely" by points over its surface. First we calculate the mean μ of these points in closed form by integrating over all the triangles. Similarly, we compute the covariance matrix of the surface points relative to μ. The three categories of features are then distinguished based on the relative lengths of the eigenvectors from this covariance matrix:

—Three nearly equal eigenvalues. (spherical)
—One eigenvalue much larger than the other two. (long ellipsoid)
—Two nearly equal eigenvalues that are much larger than the third. (flat ellipsoid)

For the long thin cylinder case, we unfold the feature into a single patch. We find the point **p** on surface with the greatest AGD. Then, we look for a point **q** that is furthest away from **p**. We find the shortest path between **p** and **q**, and proceed to split the surface along this path by duplicating all the vertices along the path. Once the surface is split along this curve, the surface become a single patch homeomorphic to a disk.

For the flat ellipsoid case, we create two patches. We first identify the direction associated with the smallest covariance eigenvalue. Then we find the two most distant surface points along this vector in opposing directions away from

the surface's center μ. We then use region growing to find the two surface Voronoi regions for these two points. Both regions are homeomorphic to a disk.

In case of a sphere, we could treat it as a flat ellipsoid and get two patches that are much like hemispheres. Unfortunately, we have found that doing so results in patches that have high stretch when flattened. Thus we use an approach that is inspired by the two identical patches of a baseball (see Figure 6, upper left). We construct these regions based on two C-shaped curves, each of which travels half the way around each of two mutually perpendicular great circles. To create these curves, we find the three pairs of antipodal points on the surface that passes through the surface center μ, along the three eigenvector directions. Call these points $x_1, x_2, y_1, y_2, z_1, z_2$. One C-curve passes through x_1, y_1, x_2, and the other goes from z_1, y_2, z_2. Using region growing, we build the surface Voronoi regions corresponding to each of these two curves. These regions are our two baseball patches. The upper right portion of Figure 6 show one of these curves and the corresponding Voronoi regions (red and green) for the igea model. Shown in the bottom row of the same figure are the normal maps of corresponding to the decomposition obtained by using two different patch unfolding methods: geometric stretch [Sander et al. 2001] (left), and Green-Lagrange tensor (Section 4.1, right). As will be discussed in Section 4.1, patch unfolding using Green-Lagrange tensor results in less overall stretch.

There are certain cases where a feature is a curved long cylinder, such as the feline's tail, but where the covariance analysis is similar to that a flat ellipsoid or a spherical surface. In this case, the center μ is situated outside the volume enclosed by the surface. Therefore, not all three pairs of antipodal points can be found. When this happens, we simply treat the surface as a long thin cylinder.

4. PATCH UNFOLDING

A class of traditional patch unfolding methods are based on approximations of conformal mapping [Eck et al. 1995; Floater 1997; Lévy et al. 2002]. If the texture coordinates of the boundary vertices are fixed, the texture coordinates of the interior vertices can be solved through closed form system. These methods are fast and stable, and the solution is unique [Lévy et al. 2002]. However, conformal mappings do not preserve areas. Regions can be stretched or compressed, causing uneven sampling rates. Sander et al. [2001] proposed a post-processing step in which the texture coordinates of the interior vertices of a patch are optimized to reduce geometric stretch, and their work inspired our own stretch optimization. We seek a definition of geometric stretch that gives a balance between conformal and area-preserving mappings.

4.1 Green-Lagrange Tensor: Balanced Stretch Metric

An *isometry* between two surfaces is a bijective mapping f that maintains distance between two metric spaces, that is, $d(f(x), f(y)) = d(x,y)$ for all points x and y in the domain. An ideal surface parameterization P would be an isometry between the surface S and its images in the texture map I, which means an everywhere even sampling is possible based on P. For most patches no isometric parameterization exists, except in the case of developable surfaces. Classical results from Riemannian Geometry states that there exists a conformal ("angle preserving") mapping between S and I. Some parameterization methods first compute a conformal parameterization for a patch, and then optimize the interior vertices based on some stretch metric [Sander et al. 2001; Lévy et al. 2002; Sheffer and De Sturler 2002]. The stretch metric of Sander (used in [Sander et al. 2001; Lévy et al. 2002]) helps balance the sampling given by the parameterization. Unfortunately, it does not distinguish between isotropic stretch and anisotropic stretch. To illustrate this point and to introduce our new balanced stretch metric, we review Sander's metric and related background.

For a triangle $T = p_1, p_2, p_3$ in the surface $S \in \mathbb{R}^3$ and its corresponding texture coordinates $U = u_1, u_2, u_3$ in $\mathbb{R}^2 =< s,t >$, the parameterization $P : U \rightarrow T$ is the unique affine mapping that maps u_i to p_i. To be more specific, let $A(r,s,t)$ be the area of the triangle formed by vertices r, s and t, then

$$P(u) = \frac{A(u,u_2,u_3)p_1 + A(u_1,u,u_3)p_2 + A(u_1,u_2,u)p_3}{A(u_1,u_2,u_3)} \tag{11}$$

The symmetric metric tensor induced by this mapping is:

$$G = \begin{pmatrix} P_s \cdot P_s & P_s \cdot P_t \\ P_s \cdot P_t & P_t \cdot P_t \end{pmatrix} = \begin{pmatrix} a & b \\ b & c \end{pmatrix} \tag{12}$$

The eigenvalues of G are:

$$\{\gamma_{max}, \gamma_{min}\} = \sqrt{\frac{(a+c) \pm \sqrt{(a-c)^2 + 4b^2}}{2}} \tag{13}$$

which represents the maximal and minimal stretch of a non-zero vector. Sander defined the average stretch metric as follows:

$$L^2(T) = \sqrt{(\gamma_{max}^2 + \gamma_{min}^2)/2} = \sqrt{(a+c)/2} \tag{14}$$

This metric is the integral of stretch of the unit vector in texture space. It is usually larger than one, and is equal to one if the mapping is a local isometry.

Equation 11 assumes that the area of the triangle equals its textural image. When we compute the global stretch, we assume the total area of the surface equals the total textural image. This means we need to add this global scale factor to each triangle.

$$\rho = \frac{\sum_{t \in S} A(t)}{\sum_{t \in S} A'(t)} \tag{15}$$

then we need to rewrite equation 11 as:

$$P(u) = \frac{A(u, u_2, u_3)p_1 + A(u_1, u, u_3)p_2 + A(u_1, u_2, u)p_3}{\rho A(u_1, u_2, u_3)} \tag{16}$$

Unfortunately under this scenario, there are other mappings between a triangle and its textural image that are not isometries for which the measure also gives the value of one. In particular, this metric cannot distinguish anisotropic stretch. For instance, all three of these tensors:

$$G = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0.5 & 0 \\ 0 & 1.5 \end{pmatrix}, \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix} \tag{17}$$

result in the same geometric stretch, but the first one is clearly the more desirable. For this reason, we suggest to use *Green-Lagrange tensor* to measure and guide patch optimization. Using Green-Lagrange tensor to measure stretch has been proposed before [Maillot et al. 1993]. However, it has not been used for patch optimization.

Use the above terminology, Green-Lagrange tensor of G is defined as:

$$T(G_t) = (a-1)^2 + 2b^2 + (c-1)^2 \tag{18}$$

It has the property that it is 0 if and only if G_t is an isometry. We therefore define the stretch as

$$E_t^2 = 2T(G) = 2((a-1)^2 + 2b^2 + (c-1)^2) = [(a-c)^2 + 4b^2] + [(a+c-2)^2] = E_{conformal}^2 + E_{area}^2 \tag{19}$$

We notice that for the tensor to be conformal, we need $a = c$ and $b = 0$. When these conditions are met, the tensor becomes a scaling of magnitude $a = c$. It is an isometry if $a = c = 1$. This metric seeks a balance between angle preservation $E_{conformal}^2$ and area preservation E_{area}^2. A triangle's mapping is an isometry if and only if $E_t = 0$. This metric distinguishes anisotropic stretch from isometry. In addition, it penalizes both undersampling and oversampling.

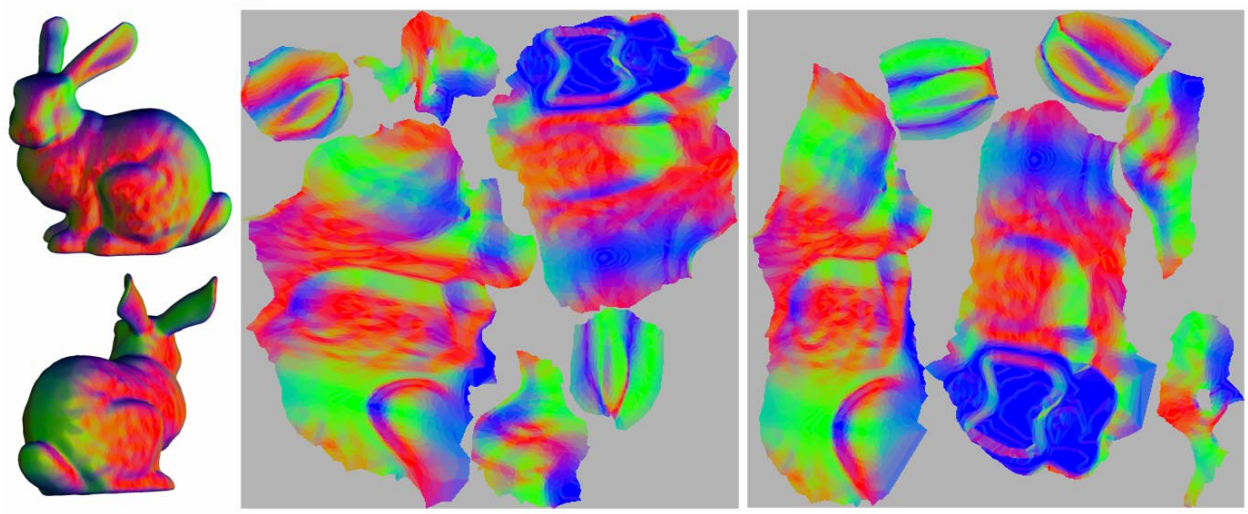

Fig. 7. This figure shows the surface parameterization of the bunny obtained by using the geometric stretch [Sander et al. 2001] (middle) and by using Green-Lagrange measure (right). Using geometric stretch cause high anisotropic stretch , especially the two largest patches. Compare the tail and the two side bumps.

However, the penalty is more severe for undersampling. This is desirable for texture mapping when a global isometry is not available. We note that Sorkine et al. [2002] devised a different stretch metric that also distinguishes anisotropic stretch from isometry. We choose not to use their metric because it uses a max function, causing it to give equal stretch values to some cases that we feel should be distinguished.

The total balanced stretch of a patch S is therefore,

$$E^2(S) = \sum_{t \in S} \{[(a_t - c_t)^2 + 4b_t^2] + [(a_t + c_t - 2)^2]\} \tag{20}$$

The ideal value $E(S)$ for a patch S is zero, meaning all triangles in the patch are mapped isometrically.

Figure 7 compares the unfolding of the bunny surface using Green-Lagrange tensor (right) with that using Sander's geometric stretch (middle). Notice on the two largest patches, unfolding with Sander's geometric stretch produces anisotropic stretch (the tail and the two side bumps). Green-Lagrange tensor performs well on all patches. Figure 6 shows the similar comparison of the igea model. Again, the normal maps shown in the bottom row are results from unfolding by using Green-Lagrange tensor (right), and the geometric stretch (left). Again, anisotropic stretch appear in the two patches. In Section 6.1 we will show Green-Lagrange tensor also performs better in terms of image fidelity, despite sometimes having lower packing efficiencies.

4.2 Boundary Optimization With Scaffold Triangles

The process of *patch optimization* refers to moving vertices in the plane to minimize a given stretch metric. Most patch optimization methods handle boundary vertices of a patch differently from interior vertices. For initial layout, boundary vertices are typically either mapped to the vertices of a convex polygon, or solved through conformal mapping. Sander et al. perform optimization by going one by one through the interior vertices and making local changes [Sander et al. 2001]. They check whether moving a given vertex along a randomly chosen line will improve the stretch of the triangles that use this vertex. We adopt a similar optimization strategy, with one modification that we describe below. During optimization there could be global foldover, in which the textural image of one boundary triangle intersects the image of another triangle which is spatially far away on the surface. Collision detection in the texture domain is

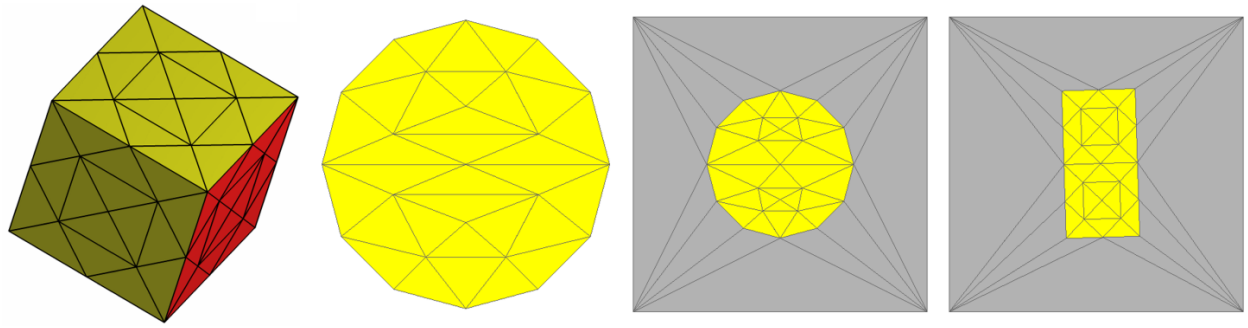

Fig. 8. This figure demonstrates the effect of scaffold triangles on patch unfolding. In the left portion, a patch on the cube is colored in yellow. In the middle-left portion, this patch is unfolded without using scaffold triangles. The middle-right image shows the unfolding of the same patch with scaffold triangles but without optimization, and the right image corresponds to unfolding with scaffold triangles and optimization. In the two rightmost images, scaffold triangles are colored in gray. When both optimization and scaffold triangles are used, the patch is unfolded entirely without stretch.

therefore needed to prevent this from happening [Sander et al. 2002].

We introduce a new optimization method that allows the boundary vertices to move freely and that avoids global foldover. First, an initial harmonic parameterization is obtained by mapping the patch boundary to a planar convex polygon and solving a linear system for the interior vertices (Figure 8, middle-left). This step is essentially the same as Sander et al [2001]. Next, we construct a "virtual boundary" (a square) in the parameterization plane that encloses the patch. The 3D coordinates of the square are assigned to be mutually different and outside the convex hull of the patch in the 3D space. As we will see next, the exact coordinates of the virtual boundary are insignificant provided they do not coincide with each other or with the patch. Scaffold triangles are used to triangulate between the original patch boundary and the virtual boundary (See Figure 8, middle-right portion). Finally, we perform patch optimization [Sander et al. 2001] on the "enlarged" patch using our metric of 4.1. There are two issues about scaffold triangles that need attention.

(1) How to define stretch for scaffold triangles.
(2) How to define and maintain their connectivity.

The first issue is handled as follows: the stretch of a scaffold triangle is defined as infinity if there is foldover, otherwise it is defined as zero. This allows a patch boundary vertex v to move within its immediate incident triangles to obtain better stretch without the need to checking for global foldover. Furthermore, the exact 3D coordinates of the virtual boundary are insignificant.

The second issue appears when the initial connectivity of scaffold triangles unnecessarily constrains the movements of boundary vertices. This is because scaffold triangles are designed to prevent global foldovers, i.e., one patch vertex "walks" onto a patch triangle other than its immediate neighboring triangles, which unfortunately include the scaffold triangles. To remedy this overly-conservative approach, we allow the re-triangulation of scaffold regions. At the end of each optimization iteration, in which all vertices (including boundary vertices) have been moved, we perform edge flips on all the edges that are adjacent to two scaffold triangles if the operations improve the triangles' aspect ratios.

The right portion of Figure 8 shows the result of optimization with scaffold triangles, which is in contrast to the middle-left portion (optimization without scaffold triangles).

The shape of the virtual boundary and the connectivity of the scaffold triangles are insignificant since they merely serve as a placeholder to allow the boundary vertices of the patch to move freely without causing global foldovers. This is very different from the work by Lee et al [2002], in which virtual boundaries are constructed as part of the springs for obtaining initial parameterization. In their work, the shape and the connectivity of the virtual boundaries directly affect the stretch of the resulting parameterization. Indeed, several layers of virtual boundaries are often required to

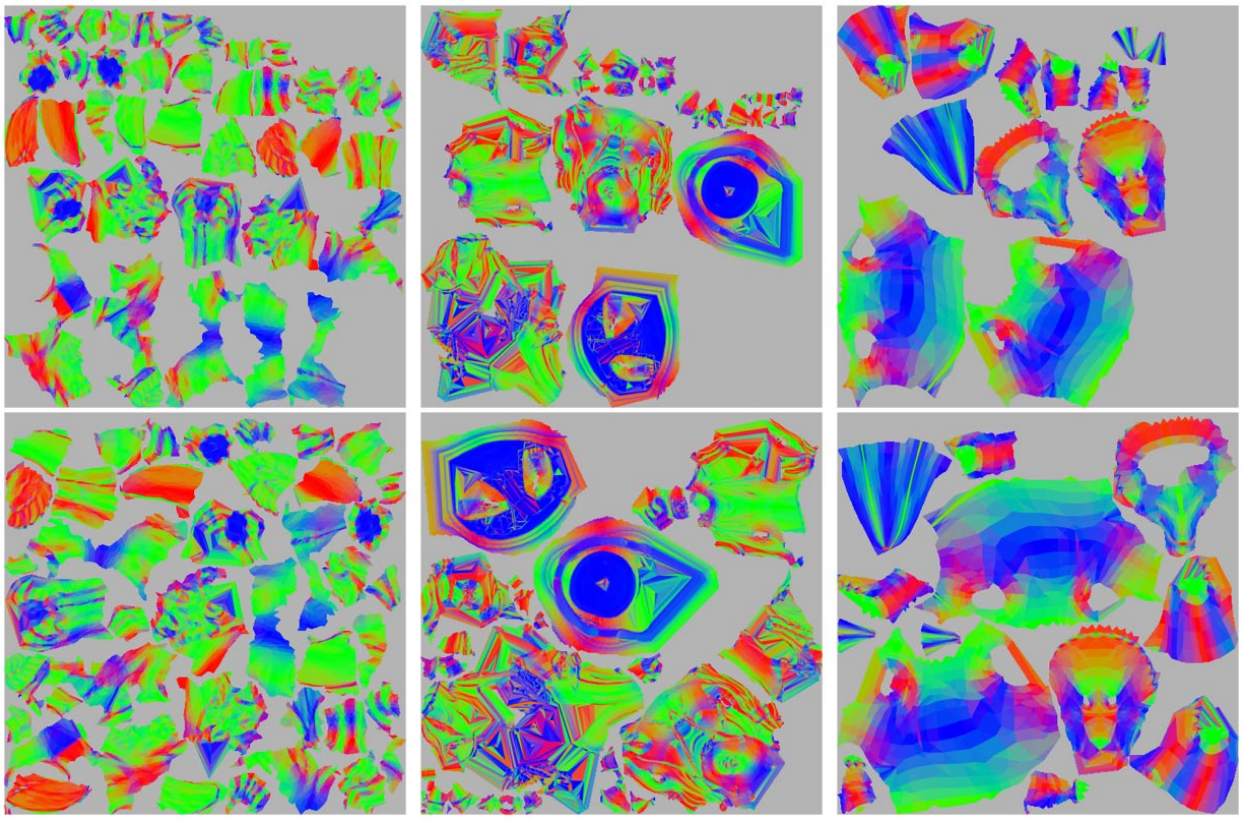

Fig. 9. This figure compares the packing results with Sander et al. [2001] algorithm (top row) and our algorithm (bottom row) for three of our test models: feline (left), buddha (middle), and dinosaur (right). Notice spaces under the "horizon" is used to pack smaller patches, and some patches are reflected diagonally to achieve tighter packing.

produce reasonable results using their method. In our work, only one layer is required.

Scaffold triangles also come from hole-filling operations that occurred during genus reduction and feature identification. We allow the vertices of the hole-filling scaffold triangles to be moved (optimized) just like the other interior vertices. These scaffold triangles do not contribute to the stretch metric, and edge flips are allowed. Several of the patches in the texture map shown in Figure 9 (right column, dinosaur) and Figure 11 (bottom row, dragon) have such holes that make use of scaffold triangles.

5. PACKING

The last step in building a surface parameterization is to pack patches inside a rectangular region, the texture map. As pointed out in [2001], this is a NP problem. Sander et al [2001] pack the bounding box of each patch by aligning them vertically and inserting one patch at a time in a horizontal-first sweeping fashion. The *horizon* of current used region is tracked. In general, horizon refers to a curve that separates the texture map into a top region and a bottom region. The bottom region contains packed patches, while new patches are inserted into the top region. New patch can not be inserted under the horizon. Lévy et al [2002] directly pack the patches without using their bounding boxes. In addition, the horizon is more carefully maintained by discretizing the rectangular region with the texel resolution. Finally, new patches can be inserted anywhere along the horizon instead of following a specific order. This results in

better packing ratios.

We propose a new packing algorithm that takes advantage of two observations. First, many unfolded patches are of elongated shapes. They can be packed either horizontally or vertically, which often results in better use of the texture space. There are eight such positions: the four 90 degree rotations plus their four reflections. Second, by packing large patches first, the gaps between these patches in the rectangular space are often large enough to contain smaller patches.

Our algorithm starts by creating the *canvas*, a rectangular grid cells with textural resolution. Each cell is initially assigned *unoccupied*. With the same resolution, we discretize the bounding box of each patch's textural image into rectangular grid cells. A cell is *occupied* if it intersects the textural image of at least one triangle in the patch. Otherwise, it is *unoccupied*. For each patch, we obtain eight variations of grid cells by the combination of reflections with respect to its vertical axis, the horizontal axis, and the diagonal.

We iteratively insert one patch into the canvas at a time in the decreasing order of the patch's textual size (area). Let N be the desired size of the final texture map. Initially, all $N \times N$ grid cells in the canvas are *unoccupied*. The first patch is placed at the lower left corner of the canvas. After one patch is inserted, some grid cells in the canvas will be *occupied*. When inserting the next patch P_i, we examine its eight variations to find the one that minimizes *wasted space* in the canvas. To be precise, let α, a $m \times n$ grid cells, be a variation for P_i. We wish to place the lower-left corner of α in the (a,b) grid cell in the canvas such that the following conditions are met.

(1) For any *occupied* grid cell (p,q) in α, the corresponding grid cell $(a+p, b+q)$ in the canvas is *unoccupied*.
(2) α minimizes $\max(a+m, b+n)$.

In other words, we wish to place the patch as close to the lower left portion of the canvas as possible. Once the best variation is chosen, we translate and scale the patch textural image to reflect its position and orientation in the canvas.

When all patches have been inserted, usually only $M \times M$ grid cells in the canvas become occupied. For all our test models, M is between one-third and one-half of N, the size of the canvas. Therefore, we perform scaling to all patches with the same factor so that $M \times M$ grid cells maps to $[0,1] \times [0,1]$.

Figure 9 shows the improvement of our packing algorithm (bottom row) over the algorithm by Sander et al [2001] (top row) for three of our test models: feline (left), buddha (middle), and dinosaur (right). Notice the space under the "horizon" is reused to pack small patches, and patches are rotated/reflected to achieve tighter packing.

6. RESULTS

We have applied our feature-based surface parameterization method to a number of test models. Figures 1 (left) and 2 (right) show some results of our feature segmentation approach. (We wish to emphasize that no real animals were harmed during our research.) Figure 6 (right) and Figure 9 (lower middle) show the normal maps of the igea and the buddha respectively. In a normal map, color (R,G,B) are used to encode unit surface normals (x,y,z) [Sander et al. 2001]. Because of many sharp creases on the igea and the buddha, we believe that patch creation methods based on surface curvature would have split the surfaces into many tiny patches, but our method was able to create large patches with little stretch. Figure 10 shows the result of our feature identification algorithm on other test models. Notice in general the created features are intuitive. For example, the horns and legs of animals are segmented from the bodies, and the buddha's stand is identified as a single feature (a flat ellipsoid).

Figure 11 shows textured models (left column) and the corresponding texture maps (right column) of the buddha (top), feline (middle), and the dragon (bottom). Table I gives the average stretch for the patches of the test models, the feature creation times, and the patch unfolding times using our method. The texture used for the buddha is a wood texture from Perlin's noise [1985]. The textures used for the feline and the dragon were created by performing example-based texture synthesis directly on the surfaces [Turk 2001; Wei and Levoy 2001].

6.1 Measuring Quality

Measuring the quality of surface parameterization is an important yet complicated issue. It has several components.

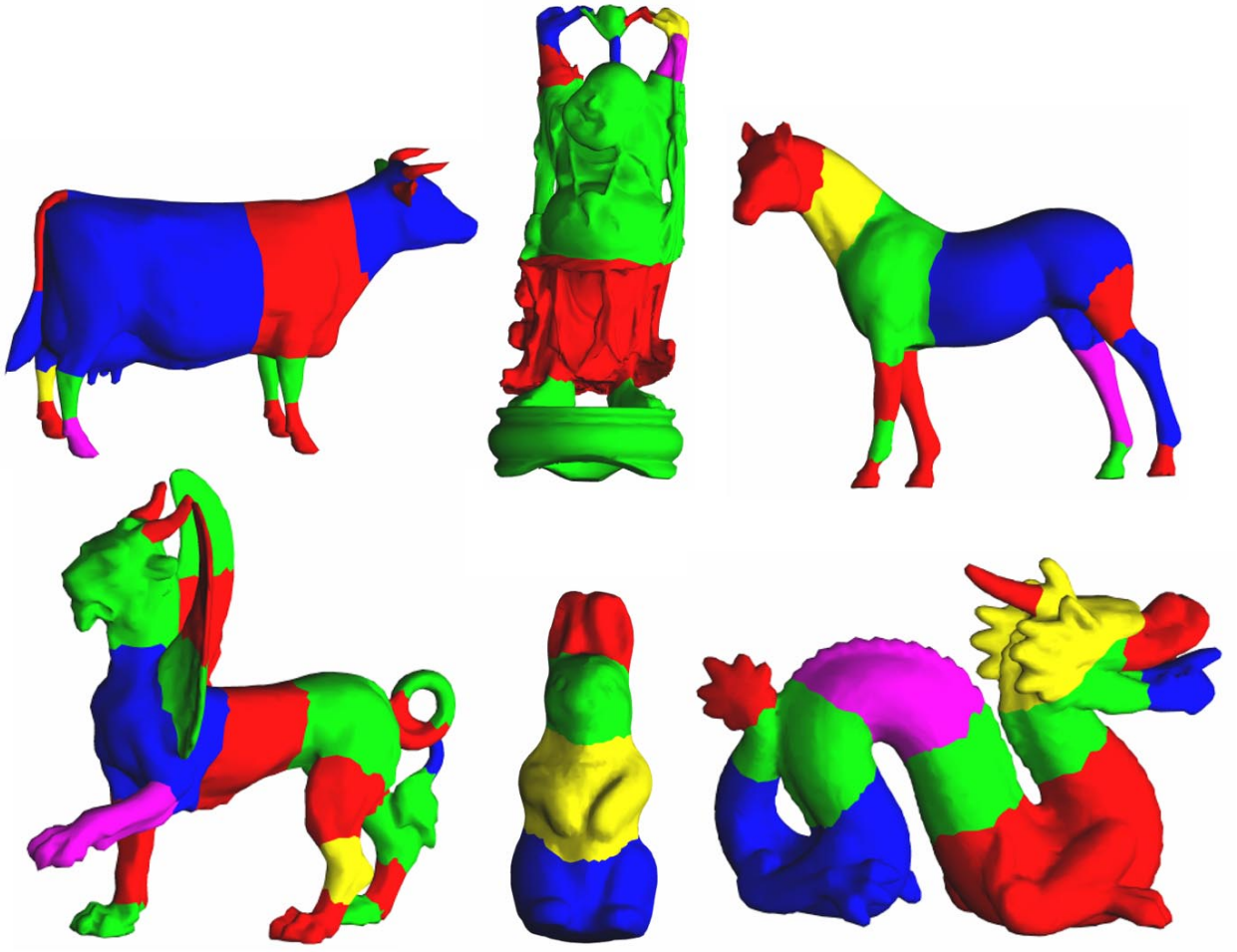

Fig. 10. This figure shows the result of our feature segmentation method on various test models. The cow, the horse and the rabbit model are genus zero surfaces. The genus of the original models for the dragon, the happy buddha, and the feline are one, six, and two, respectively.

(1) Stretch: affects the sampling rate across the surface.
(2) Seams: causes discontinuity across the patch boundaries.
(3) Smoothness: cause sharp change in the sampling rate across interior patch edges.
(4) Packing ratios: determines the efficiency of the use of the texture map.

When evaluating a surface parameterization method, it is not clear how these components should be combined to give indication to the quality of the resulting map. On the other hand, for texture mapping applications, the quality of a surface parameterization should reflect “image fidelity”, i.e., the faithfulness of the images produced using texture maps to the images in which surface signals are directly computed. We propose to use such an image-based metric, which draws inspiration from the work by by Lindstrom and Turk for image-driven mesh simplification [Lindstrom and Turk 2000].

To be more specific, we first select a continuous and smooth surface signal. Then we compute this signal and store the result in a texture map based on the surface parameterization. Finally, we render the surface from many view

Fig. 11. This figure shows the parameterization of three models using our feature-based algorithm. The rows from top to bottom are: the buddha, the feline, and the dragon in the textured models, and the texture layouts (512×512). The genus of the original models for the buddha, the feline, and the dragon are six, two, and one, respectively.

model name	# polygons	# patches	stretch (Green-Lagrange)	feature time	unfold time
buddha	20,000	28	1.56	6:32	27:29
bunny	10,000	6	0.23	1:07	8:28
cow	10,524	29	0.28	2:15	4:01
dinosaur	10,636	14	0.25	1:37	5:53
dragon	20,000	24	0.83	3:00	16:23
feline	10,000	41	0.22	2:31	2:32
horse	10,000	27	0.22	1:30	3:21
igea	10,000	2	0.17	0:11	11:38
rabbit	10,000	8	0.24	0:53	4:50

Table I. Average stretch (measured in Green-Lagrange) and timing results (minutes:seconds) for feature segmentation and patch unfolding. Times are for a 2.4 GHz PC.

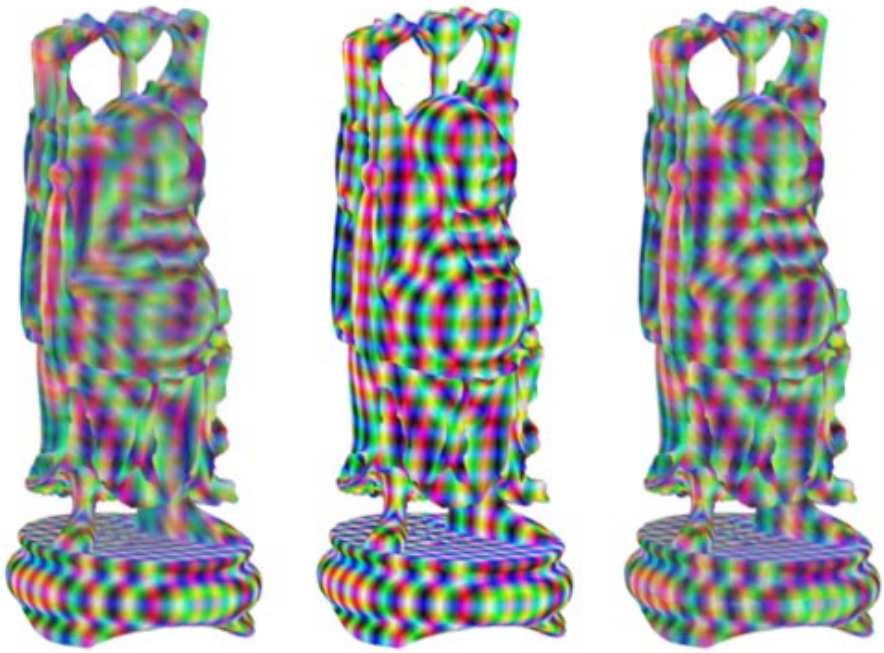

Fig. 12. This figure shows the visual comparisons between patch unfolding with the geometric stretch [Sander et al. 2001] (left) and with Green-Lagrange stretch tensor (right) for the buddha model. The 3D texture used here is described in Section 6.1. The middle image is obtained by directly computing the texture on surface. Compare the signal on buddha's face, body and feet. Unfolding with the geometric stretch [Sander et al. 2001] tend to create uneven sampling rates.

points using the texture map and compare the image differences with respect to the true surface signals. In practice, we choose 20 orthographic view points that are the vertices of a surrounding dodecahedron. Let M_0 be the surface with the signal directly computed, and M_i be the textured surface with the texture size of $2^i \times 2^i$. The RMS "image" error between the images is calculated as:

Comparison for patch unfolding with different optimization metrics (top row using geometric stretch [Sander et al. 2001], bottom row using Green-Lagrange tensor)					
	Stretch measured in geometric stretch	Stretch measured in Green-Lagrange	Packing Ratio	Image Error 128×128	Image Error 256×256
buddha	1.27	26.80	0.67	5.61%	2.99%
	1.18	1.56	0.68	5.11%	2.69%
bunny	1.13	3.92	0.60	5.47%	2.86%
	1.02	0.23	0.65	4.62%	2.53%
cow	1.11	3.07	0.73	3.72%	2.33%
	1.03	0.28	0.65	3.65%	2.28%
dinosaur	1.07	1.55	0.59	3.20%	2.03%
	1.03	0.25	0.66	2.75%	1.84%
dragon	1.26	13.78	0.67	6.23%	3.18%
	1.13	0.83	0.67	5.48%	2.90%
feline	1.10	1.73	0.66	3.43%	2.00%
	1.02	0.22	0.64	3.20%	1.94%
horse	1.09	1.65	0.67	3.10%	2.00%
	1.03	0.22	0.66	2.99%	2.01%
igea	1.10	2.99	0.59	6.07%	3.31%
	1.02	0.17	0.66	5.09%	3.05%
rabbit	1.12	3.00	0.68	4.43%	2.55%
	1.03	0.24	0.65	4.09%	2.42%

Table II. This tables compares two stretch metrics for guiding Sander style vertex optimization. The top row in each data cell are the results from using the geometric stretch, and the bottom rows are the results from using Green-Lagrange tensor, (Section 4.1). For a comprehensive comparison, three measurements are provided: average stretch (the first two columns), packing ratio (third column), and image-based error metric (Section 6, the last two columns). For all nine test models, optimization with Green-Lagrange tensor results in lower average stretch (either measured using geometric stretch or Green-Lagrange tensor). Consequently, optimization with Green-Lagrange tensor results in lower image errors for all models except for the horse for 256×256 texture map, which is a direct result of lower packing ratio.

$$RMS(M_i, M_0) = \sqrt{\sum_{n=1}^{20} D_i^n} \tag{21}$$

Here, D_i^n is the squared sum of pixel-wise intensity difference between the n-th image of M_i and M_0.

One possible ideal surface signal can be obtained by first spreading a set of evenly-spaced points on the surface and building a smooth function that uses these points as the bases. However, it can be time consuming to produce such a function. On the other hand, we notice the $3D$ checkerboard pattern has the nice property that it is easy to compute and the biggest differential in frequencies in all directions is bounded. Although not perfect, it is nonetheless a good starting point. To make the signal continuous, we replace each "box" section with a "hat". The frequency in each main axial direction is the same. In practice, we use $1/16$ of the maximum side of the bounding box of the surface as the frequency.

Table II compares two unfolding methods, optimization with geometric stretch metric [Sander et al. 2001] and our metric for nine test models. Notice optimization with our metric produces lower stretch for all the test models. Furthermore, despite sometimes having lower packing ratios, optimization with our metric produces lower image errors for all the test cases except for the horse model at the size of 256×256. Figure 12 shows the visual comparison between the ideal signal (middle), the textured model using optimization with the geometric stretch [Sander et al. 2001] (left) and our metric (right) for the buddha model with the texture map of size 128×128. Notice the different level of blurring in the left image (front body and base) due to uneven sampling rate. This phenomenon is less noticeable in the right image that uses our approach.

7. CONCLUSION AND FUTURE WORK

In this paper, we present an entirely automatic feature-based surface parameterization method in which manifold surfaces are divided into feature regions for patch creation. We also introduce the use of Green-Lagrange tensor to guide the vertex optimization for patch unfolding. In addition, we use scaffold triangles to allow the boundary vertices of a patch to be optimized. Furthermore, we present a new patch packing algorithm that increase the packing ratio. Although they were developed with texture mapping in mind, we think our surface segmentation and genus reduction methods might also be useful for other applications. Finally, we propose an image-based quality measure for surface parameterization techniques.

Many additional topics in this area are of interest to us. First, we would like to see whether AGD can be used to decide the region of separation. Since AGD is less noisy and is intrinsic to the surface, we expect that it contains more useful information.

Second, we are still looking for other Morse functions for surface segmentation. Our metric has been chiefly based on surface distance, which is intrinsic to the surface. Are there other functions which combines both intrinsic and extrinsic properties of the surfaces that might result in even better segmentation? For instance, is there a surface function that help find a separation curve for the feline's wing at exactly where human would place it.

Surface parameterization is important for many applications, one of which is surface visualization. A complex surface often contains interiors and concavities, which are difficult to see from the outside viewpoints. We would like to investigate the use of parameterization for surface exploration purposes, giving the user the ability to navigate, orient and focus.

ACKNOWLEDGMENTS

We would like to thank the following people and groups for the 3D models they provided: Zoë Wood and Peter Schröder, Mark Levoy and the Stanford Graphics Group, Andrzej Szymczak. We also appreciate the discussions with Jarek Rossignac and Andrzej Szymcazak.

This work is funded by NSF grant ACI 0083836 and NSF grant 0107396.

REFERENCES

ALLIEZ, P., MEYER, M., AND DESBRUN, M. 2002. Interactive Geometry Remeshing. *ACM Transactions on Graphics (SIGGRAPH 2002) 21,* 3, 347–354.

BENSON, D., AND DAVIS, J. 2002. Octree textures. *ACM Transactions on Graphics (SIGGRAPH 2002) 21,* 3, 785–790.

CARR, N. A., AND HART, J. C. 2002. Mesh Atlases for Real-Time Procedural Solid Texturing. *ACM Transactions on Graphics 21,* 2, 106–131.

CIGNONI, P., MONTANI, C., ROCCHINI, C., AND SCOPIGNO, R. 1998. A General Method for Recovering Attribute Values on Simplified Meshes. *IEEE Visualization '98*, 59–66.

DEBRY, D. (GRUE), GIBBS, J., PETTY, D. D., AND ROBINS, N. 2002. Painting and rendering textures on unparameterized models. *ACM Transactions on Graphics (SIGGRAPH 2002) 21,* 3, 763–768.

ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution Analysis of Arbitrary Meshes. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1995)*, 173–182.

ERICKSON, J. AND HAR-PELED, S. 2002. Optimally Cutting a Surface into a Disk. *Symposium on Computational Geometry*, 244–253, Barcelona, Spain, (June).

FLOATER, M. S. 1997. Parameterization and Smooth Approximation of Surface Triangulations. *CAGD 14*, 3 (July), 231–250.

GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1996)*, 171–180.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry Images. *ACM Transactions on Graphics (SIGGRAPH 2002) 21*, 3, 355–361.

HANRAHAN, P., AND HAEBERLI, P. E. 1990. Direct WYSIWYG Painting and Texturing on 3D Shapes. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1990)*, 215–223.

HILAGA, M., SHINAGAWA, Y., KOHMURA, T., AND KUNII, T. L. 2001. Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, 203–212.

HORMANN, K., AND GREINER, G. 1999. MIPS: An Efficient Global Parameterization Method. In *Curve and Surface Design: Saint-Malo 1999*, edited by Laurent, Sablonnière and Schumaker, Vanderbilt University Press 2000, 153–162.

KACZYNSKI, T., MISCHAIKOW, K., AND MROZEK, M 2001. Computing Homology. In *preprint, Department of Mathematics, Georgia Tech, www.math.gatech.edu/ mischaik/papers/atmcs.ps*, October 2001

KIMMEL, R., AND SETHIAN, J. A. 1998. Computing Geodesic Paths on Manifolds. *Proceedings of National Academy of Sciences 95*, 15, 8431-8435, (July) 1998.

LAZARUS, F., POCCHIOLA, M., VEGTER, G., AND VEROUST, A. 2001. Computing a Canonical Polygonal Schema of an Orientable Triangulated Surface. *17th ACM Symposium on Computational Geometry*, 80-89, (June).

LEE, A., SWELDENS, W., SCHRÖDER, P., COSWAR, L., AND DOBKIN, D. 1998. MAPS: Multiresolution Adaptive Parameterization of Surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1998)*, 95–104.

LEE, Y., KIM, H. S., AND LEE, S. 2002. Mesh parameterization with a virtual boundary. *Computers & Graphics (Special Issue of the 3rd Israel-Korea Binational Conf. on Geometric Modeling and Computer Graphics) 26*, 5, 677–686.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Transactions on Graphics (SIGGRAPH 2002) 21*, 3, 362–371.

LINDSTROM, P., AND TURK, G. 2000. Image-Driven Simplification. *ACM Transactions on Graphics 19*, 3, 204–241.

MAILLOT, J., YAMIA, H., AND VERROUST, A. 1993. Interactive Texture Mapping. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1993)*, 27–34.

MILNOR, J. 1963. Morse Theory. In *Annals of Mathematical Studies. Princeton University Press 51*, Princeton, NJ 1963.

PERLIN, K., 1985. An Image Synthesizer. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1985)*, 287–296.

PIPONI, D., AND BORSHUKOV, G. 2000. Seamless Texture Mapping of Subdivision Surfaces by Model Pelting and Texture Blending. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2000)*, 471–478.

SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. 2001. Texture Mapping Progressive Meshes. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, 409–416.

SANDER, P. V., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2002. Signal-Specialized Parameterization. *Proc. 13th Eurographics Workshop on Rendering 2002*, 87–100.

SHEFFER, A., AND HART, J. 2002. Seamster: Inconspicuous Low-Distortion Texture Seam Layout. *IEEE Visualization 2002*, 291–298.

SHEFFER, A., AND DE STURLER, E. 2002. Smoothing an Overlay Grid to Minimize Linear Distortion in Texture Mapping. *ACM Transactions on Graphics 21*, 4, 874–890.

SORKINE, O., COHEN-OR, D., GOLDENTHAL, R., AND LISCHINSKI, D. 2002. Bounded-distortion Piecewise Mesh Parameterization. *IEEE Visualization 2002*, 355–362.

TURK, G., 2001. Texture Synthesis on Surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, 347–354.

DE VERDIERE, E. C., AND LAZARUS, F. 2002. Optimal Polygonal Schema on an Orientable Surface. *18th European Workshop on Computational Geometry*, Warszawa, Poland, (April), 627–636.

WEI, L. Y., AND LEVOY, M. 2001. Texture Synthesis over Arbitrary Manifold Surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, 355–360.

WOOD, Z., HOPPE, H., DESBRUN, M., AND SCHRÖDER, P. 2002. Isosurface Topology Simplification. *Technical Report*.